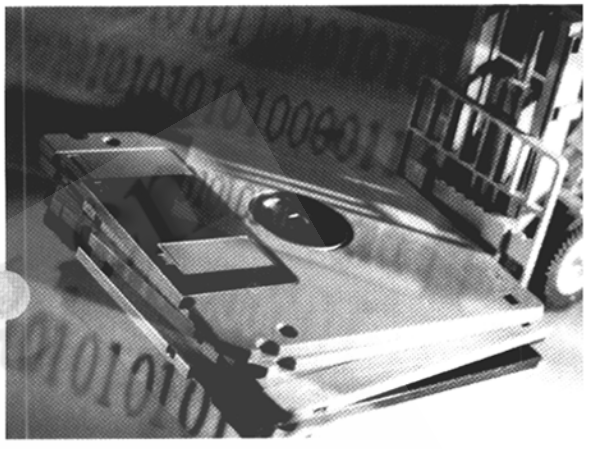
如何开发高质量低成本的软件

冯宗文 (浙江大学西溪校区计算中心 310028)



摘要：该文从软件工程的角度出发，提出要开发高质量低成本的软件惟有提高软件的可靠性和可维护性，以及相应的技术手段和保障；并提出在软件项目中引入监理，可以降低项目风险，提高软件质量，降低软件开发成本。

关键词：软件 高质量 低成本 开发

1 引言

随着计算机不断地发展与普及，软件在计算机应用中起着越来越重要的作用，正在运行使用的计算机软件的数量以惊人的速度急剧膨胀，但是在软件的开发过程中，人们又遇到了许许多多的困难，突出的问题是软件质量不高而开发成本居高不下。这一问题已经成为当今软件业发展的一大瓶颈问题。开发一个软件系统，就是一项投资，目的是期望在将来获得更大的经济效益，无论其效益是直接的或是间接的。

2 影响软件开发质量和成本的主要因素

影响软件开发质量和成本的因素有很多，诸如：软件的功能性，软件的可靠性，软件的可维护性，效率，风险，软件的适应性，软件的安全性，软件的可移植性等等。

在诸多因素中，软件的可靠性、软件的可维护性是最主要的，它们的好坏会直接或间接地影响其他因素。当然软件的功能性这个前提也不可忽视，由于它表现为系统在完成预定应该完成的功能时令人满意的程度。如果系统不能满足用户的需要，或与用户的需求不一致，这样的软件就无质量可言，也就无可靠性、可维护性可谈了。因此软件开发应首先保证软件的功能性，在此前提下提高软件的可靠性和可维护性。

软件可靠性是一个软件按照用户需求和设计者的相应设计，执行其功能的正确程度，包含了正确性和健壮性两个方面。如果一个软件不能对正确的原始数据运行得到正确的结果；如果支撑环境（如硬件）发生故障或原始数据有错误，造成软件不能正常运行，甚至造成灾难性的后果，显然其他一切都无从谈起。提高了软件的可靠性，就能使软件在正常情况下正确工作；在意外发生时，能采取有利措施及时处理，控制事故的蔓延，防止信息的丢失，并使系统恢复原形。

软件可维护性指阅读程序时，易于理解的程度；运行中发现其中的错误或缺陷、准备加强其功能、改善其性能时，需对程序进行修改、变更的难易程度。按B.W.Boehm的定义，软件的可维护性包括软件的可理解性、可测试性和可修改性三个方面。如果没有清晰的程序结构，没有采用模块化、结构化；如果编码中使用了令人难以琢磨的程序技巧，没有足够的注释，这样的软件程序又如何让人阅读、理解？如何进行调试、修改？如何去扩充新的模块、增加新的功能？提高了软件的可理解性，就能使人们容易理解软件的结构、接口、功能和内部过程，使测试、诊断容易进行；提高了软件的可测试性，就能使软件的正确性易于论证，便于实现校正性维护；提高了软件的可修改性，就能使测试中发现的错误易于纠正，减少因修改而引入的错误，就能便于扩充新的模块，增加程序功能。

有统计资料表明，在软件开发、运行及维护的整个过程中，维护阶段的花费大约占软件生存期花费的55%——70%。因此要降低开发成本，首先应该从降低软件的维护成本着手。而要降低维护成本、提高软件质量，又必须以软件的高可靠性和高可维护性为基础。软件的可靠性和可维护性不仅与软件开发成本密切相关，还是评价软件质量优劣的重要指标。

3 可采取的措施

3.1 模块化与抽象化技术

利用模块化技术可以将复杂问题分解为若干个容易解决的小问题（模块），原问题也就容易解决了。模块划分时，不宜过小、数量不宜过多，否则将造成模块之间接口工作量的增加；也不宜太大，否则会给测试和维护带来困难。采用模块化技术可以使软件结构清晰，也使软件容易设计、容易阅读、容易理解，提高了软件的可理解性；采用模块化技术能将错误或变动局限于有关的几个模块及他们的接口之间，使软件容易测试和调试，并能防止错误的蔓延，提高了软件的可靠性、可修改性。

人们在认识复杂现象时，抽象是最有力的思维工具。抽象就是抽出事物的本质特性而暂时不考虑它们的细节。对于大型复杂系统可以通过层次方式去构造分析，先用一些高级的抽象概念来构造，这些高级概念又可以用一些较低级的概念来构造，如此下去，直至最底层的具体元素。软件定义、开发过程中的每个步骤都是抽象层次的一次精化，自顶向下由抽象到具体的方式，简化了软件的设计与实现，提高了软件的可理解性和可测试性，并使软件更容易维护。

3.2 信息隐蔽与局部化技术

大型软件系统在整个软件生命周期中都要经受多次修改，在划分模块时就应该采取相应措施，使修改造成的不利影响尽可能局限于一个或少数几个模块内。为此Parnas于1972年提出了信息隐蔽原理。

信息隐蔽是指模块设计应该使包含在模块内的信息（过程和数据）是其他模块不能访问的。也就是说，信息隐蔽技术将某个因素隔离在一个模块的内部，当这个因素发生变化时，这种变化不会传播到该模块之外。对于这样构造的软件系统，在将来某些因素发生变化需要修改维护时，只要修改一个模块就够了，其他模块不会受到影响。

局部化是指把一些关系密切的软件元素物理地放得彼此靠近。局部化与信息隐蔽是密切相关的，它有助于信息隐蔽，使独立的模块间仅仅交换为完成软件功能而必须交换的信息。因为大部分的数据和过程对软件的其他部分是隐蔽的，所以即使在修改期间由于疏忽而引入的错误，也不大可能波及到软件的其他部分。

由于修改极易引起错误，如果修改的影响范围越小，因修改而引起错误的可能性就越小。信息隐蔽技术不仅大大提高了软件的可维护性，减少测试、维护期间修改软件时所需的工作量，还提高了软件的可靠性。

3.3 软件结构与意外事故的防范

软件结构对可靠性、可维护性的影响很大，不良的软件结构容易隐含错误，尤其在修改时更容易造成错误。为提高可靠性、可维护性，最基本的措施之一是使软件结构简单清晰。

好的软件结构应使每个模块完成一个相对独立的特定功能，并与其他模块之间的接口简单。独立的模块功能单一、接口简单，使得软件容易开发，特别是对大型软件来说易于分工合作共同完成；能使设计和代码修改引起的副作用被充分限制，错误不易蔓延，易于软件的测试和维护。

对于可能发生的意外事故，例如硬件意外故障，或者操作人员输入了非法信息等等，如果在设计时没有充分考虑防范措施，那么这样的系统将是非常脆弱、难以维护的，使用这样的系统必将遭受到重大损失。

Parnas主张设计时就应该预计将来可能发生的各种意外事件，并采取措施，提高系统的可靠性。针对硬件可能发生的故障，接近硬件的模块应该检测硬件的行为，以便及时发现硬件故障，并尽量避免软件的重复检测；针对操作人员可能发生的失误（或故意破坏），负责接受输入的模块应该对输入的数据进行合理性检查，辨认非法、越权的操作请求，并提供纠正错误的手段；针对软件本身可能存在的错误，甚至重大、致命的错误，在模块之间应该加强检查，防止错误的蔓延。

3.4 程序设计语言与程序设计风格

开发软件系统时，必须选择一种程序设计语言来实现这个系统，合适的语言可以使编码减少困难，减少程序的测试量，得到更容易阅读、更容易维护的程序。实践表明，高级程序设计语言较汇编语言有许多的优点，除了一些很特殊的应用领域，或者在大型系统中执行程序时间非常关键的地方，需要用汇编语言编程之外，其他程序一般都使用高级语言编程。

高级语言有其自身的特性，在实际选择编程语言时，除了要考虑理论上的标准，还要考虑使用上的各种限制，例如系统用户的要求、可使用的编译程序、可得到的软件工具、工程的规模、程序员掌握语言的情况、软件可移植性要求、软件的应用领域等，综合各方面的因素，选择合适的程序设计语言。

在程序编写中还有一个十分重要但容易被忽视的问题，那就是程序设计风格。好的程序设计语言有助于写出可靠而易维护的好程序，但使用不当，工具再好还是不能

达到预期的效果。编写出来的程序不仅要严格遵循所用程序设计语言的语法规定，写出没有语法错误的程序，同时还应该遵循某些程序设计风格的要求。编写程序时，应该注意程序内部良好的文档资料（如符号名的命名、注释行的使用）、有规律的数据说明格式、清晰简洁的语句结构和输入／输出格式、错落有序的层次结构（如空行和缩格的使用）、不使用任何编程技巧、避免使用过于相似的符号名、尽量用显式说明来定义所有的变量、避免不必要的GOTO语句、注意浮点运算的误差、尽量减少使用语句标号等。只有这样，才能使写出的程序具有良好的可读性和可理解性，才能有助于及时发现和改正程序中的错误，提高软件的可靠性和可维护性。

3.5 完善文档的重要性

软件文档在软件开发中占有突出的地位，特别是在维护阶段，文档比可执行的源程序代码更为重要。完善的文档是软件可移植性、可用性和可维护性的基本保证。文档包括用户文档和系统文档，用户文档应该能使用户获得对系统准确的初步印象，使用户能够方便地根据需要阅读有关的内容；系统文档从问题定义、需求分析到验收测试计划等一系列和系统实现有关的文档。对理解程序、维护程序来说，系统设计、实现和测试的文档都是极其重要的。不管是哪一类文档资料都必须和程序代码同时维护，只有和程序代码完全一致的文档才有真正的价值。

3.6 软件复用技术

软件复用又称软件重用，是指在构造新的软件系统的过程中，对已存在的软件人工制品的使用技术。软件人工制品可以是源程序代码段、设计结构、模块级实现结构、规范说明、文档、转换程序等。但是运行中的重复不属于软件的复用，如执行期间的重复调用，程序重复运行，分布处理中的程序段拷贝等，都不属于软件复用。

大量的实践表明，只有当软件设计以可重用性为设计目标时，才可得到可重用软件。可重用的软件成分通常包括模块、函数、过程、抽象数据、模式、子模式、进程等。实现软件复用的方式是组装和生成技术。组装是把可重用软件成分组装成新的软件，通过子程序库技术、基于面向对象的软件集成技术、共享接口设计、嵌套函数调用等方法来实现，组装对软件复用成分通常不作修改或只作很少的修改。生成技术是对模式的复用，由软件生成器通过替换特定参数，生成抽象软件成分来实现。

软件复用技术利用已有的、成熟的、正确的软件成分来构造新的软件，可以大大缩减软件开发所需的人力、物力、财力和时间，还能提高软件的可靠性和可维护性，降低软件开发成本。

3.7 面向对象技术

面向对象方法尽可能模拟人类习惯的思维方式，使开发软件的方法与过程尽可能接近人类认识世界解决问题的方式与过程，也就是使描述问题的问题空间与实现解法的解空间在结构上尽可能一致。这样的软件系统由对象组成，对象是能完整地反映问题本质的实体。

面向对象方法强调把问题域的概念直接映射到对象以及对象之间的接口，符合人们通常的思维方式；它从分析到设计再到编码都采用一致的模型表示，后一阶段可以直接复用前一阶段的工作成果，大大减少了工作量；它以对象为中心构造软件系统，对象是将属性和操作封装为一体，当外部功能发生变化时，能保持对象结构的相对稳定，使改动仅局限于一个对象的内部，减少改动所引起的系统波动效应；对象具有继承性和封装性，它支持软件复用，并且易于扩充，能较好地适应复杂的大型系统不断发展和变化的需求。

面向对象的系统开发方法提供了一种描述现实问题的方法，便于用软件方法更好地解决实际问题，它将信息和处理过程模块化，把数据对象和过程处理联系起来，形成一个相对独立、相对稳定的整体，而不仅仅是处理过程。它建立了数据抽象、信息隐蔽和模块化等三个主要的软件设计概念，用识别问题空间中的对象建立数据抽象；通过定义操作来描述软件模块、建立软件结构，根据建立使用对象的机制（即消息）来描述对象的界面。

使用面向对象技术，能够开发出稳定性好、可重用性好、可维护性好的软件，降低软件开发成本。

3.8 加强和完善软件的测试

无论怎样强调软件测试的重要性以及对软件可靠性的影响都不过分；无论怎样力求在开发的每个阶段加强审查，尽早发现并纠正差错，但仍有“漏网之鱼”。如果软件在正式运行之前没有发现并纠正其中的大部分差错，那么这些差错在以后的运行过程中迟早会暴露出来，到那时不仅改正的代价更高，而且还会造成很恶劣的后果。

测试的目的在于发现软件中的错误，发现错误之后必须诊断并改正错误。大量统计资料表明，软件测试的工作量占总工作量的40%以上，其成本可能是其他开发步骤总成本的三——五倍，测试的结果也是分析软件可靠性的重要依据。因此，必须高度重视并加强完善软件的测试，提高软件的可靠性，降低维护成本。

测试的关键是测试用例的设计，其方法可以分为：白盒测试和黑盒测试。白盒测试根据程序内部逻辑来设计测试用例，检查程序中的逻辑通路是否都按预定的要求正确地工作。黑盒测试根据规格说明书规定的功能来设计测试用例，检查程序的功能是否符合规格说明的要求。由于不可能进行穷尽测试，因此必须仔细设计测试方案，力争用尽可能少的测试发现尽可能多的错误。

3.9 引入软件项目监理

实行项目监理制是国际上确保工程项目质量和进度的一种通行惯例。1986年，我国开始探索、引入项目监理制度并在工程建设中予以应用，收到明显效果。在软件项目、特别是大型软件项目开发建设中引入监理，可以极大地降低项目风险，提高软件质量，降低软件开发成本。

首先，监理非常熟悉信息技术和信息产品，有丰富的、成功的软件项目建设的经验，能够向用户提供大量关于软件开发技术（工具、方法）的信息和信息处理能力的支持，能够对软件项目中所采用的方案和技术进行正确评价和选择，并能对开发方提供的各种证明材料、开发工具等市场信号进行辨析，从而改变了用户在与开发方对策过程中不利选择的地位，降低了因开发方隐蔽信息而造成的项目风险和成本估算问题。

其次，在软件项目的实施过程中，监理可以根据自己的经验判断开发方是否偏离了用户的实际需求，是否简化了系统的功能模块而降低性能要求，是否隐含系统缺陷或安全问题等。这时，监理可以为用户提供支持，使用户能了解开发方是否有“偷工减料”或是“偷懒”的行为，从而改变了用户在与开发方对策中的地位，降低了因开发方的隐蔽行动而造成的软件质量问题。

4 结束语

虽然影响软件系统质量的因素有很多，既牵涉到软件开发的各个方面，也和实际开发人员水平的高低有着密不可分的关系。但总可以找到一些措施或规程，尽可能规范软件开发的运作，使软件开发行为标准化、透明化并可监督，减少实际开发人员的人为依赖，在“流水线”上生产出质量更高而成本更低的产品。■

参考文献

1 陈增荣，软件开发方法，复旦大学出版社。

2 钟珞、徐宝文，计算机软件方法学，中国铁道出版社。

3 苏金树等，计算机辅助软件工程CASE基础与技术，人民邮电出版社。

4 王立福等，软件工程——技术、方法与环境，北京大学出版社。

5 张海藩，软件工程导论（第二版），清华大学出版社。

6 左美云，信息项目监理的管理学分析，计算机系统应用，2000(5)。